TÉCNICO LISBOA

Highly Dependable Systems

MEIC-A

HDS Ledger - 1st Stage

Authors:

David Belchior (95550) davidbelchior@tecnico.ulisboa.pt
Diogo Santos (95562) diogosilvasantos@tecnico.ulisboa.pt
Vasco Correia (94188) vasco.cardoso.correia@tecnico.ulisboa.pt

Group 4

2022/2023 – 2nd Semester, P3

Contents

1 Objective

This stage of the project aims to implement a primitive version of the Istanbul BFT consensus algorithm [4] as the structure for a permission-based, Byzantine fault-tolerant (**BFT**) blockchain protocol, where both clients and blockchain nodes know each other via a public key infrastructure (**PKI**).

2 The building blocks

2.1 Channels

Java's **java.net** package already provides **UDP sockets**, which are equivalent to the fair-loss links.

2.2 Authenticated Perfect Links

From the fair-loss links abstraction we build Authenticated Perfect Links according to the implementation discussed in class and added an optimization to avoid the endless loop.

Each message is sent with the correspondent digital signature (DSA) that is computed by encrypting the digest of the original message with the sender's private key (obtained using the RSA algorithm).

The message is then stored at the destination (to avoid duplicates) and a ACK message is sent to notify the sender that the message has been received.

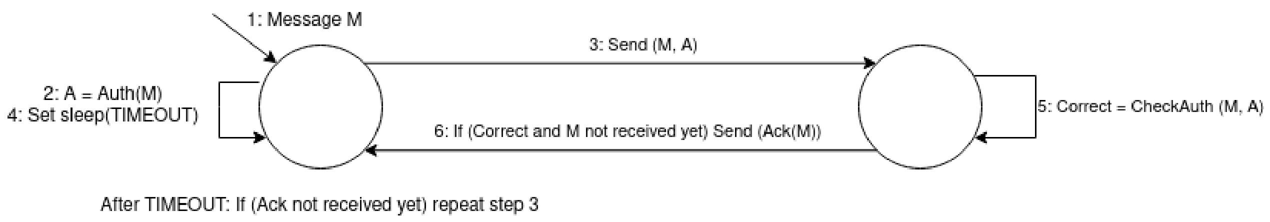
1: Message M
3: Send (M, A)
2: A = Auth(M)
4: Set sleep(TIMEOUT)
5: Correct = CheckAuth (M, A)
6: If (Correct and M not received yet) Send (Ack(M))
After TIMEOUT: If (Ack not received yet) repeat step 3

Figure 1: Visual representation of authenticated perfect links

2.3 Consensus

Consensus is achieved by a set of nodes that follow the QBFT algorithm. Each node is listening on a given port for requests from other nodes and on another port for requests from the client. The nodes communicate among them using a broadcast primitive and with the client using a send primitive all based on the Authenticated Perfect Link abstraction.

2.4 Application

The application is a simple client that reads commands from the terminal and calls an API from the library which is responsible for sending and receiving messages from the consensus layer abstraction and for storing the known blockchain state. The latter prevents the nodes from having to send all the blockchain content on every request.

The proposed value by the client is also signed to prevent any byzantine node from changing the originnal value during the algorithm.

3 Architecture of the system

We structured our implementation using an architecture similar to Google's gRPC [1], with multiple services doing asynchronous work. This involves a service responsible for receiving requests from the clients and responding to them, and a service running on every node, implementing the IBFT consensus algorithm [4].

Data marshalling and unmarshalling is done using Google's GSON [2] library.

To help lauching all the instances and testing, we have a Python script that launches multiple terminal windows with either a node or a client running.

4 Testing

For our tests, we implemented two types of configuration files, one containing clients and other containing blockchain nodes. To test the system we have multiple node configuration files where nodes can have byzantine behavior, such as:

- **Drop packets**: All received packets are dropped;
- **Bad consensus**: Attempt to start consensus, to test if other nodes only accept messages from the actual leader;
- **Fake leader**: Send messages pretending to be the leader - Because other nodes fail to verify the signature, they will not reply with ACK meaning that this node will be stuck sending messages forever;
- **Fake value**: Send messages with a single fake value into the consensus algorithm;
- **Bad broadcast**: Send messages with random values for different nodes.

Since there is always a quorum of correct nodes, all consensus instances should run successfully.

All tests were ran with no problems for each of the scenarios, proving the availability, reliability, safety and integrity attributes of dependability [3] of the protocol and our system are ensured.

5 Future improvements

Despite the success of our project so far, we put some thought into problems that may arise if we remove some of our pre-conditions:

- **If clients don't know who's the leader**: We came up with two possibilities:
  - Ask the nodes who is the leader (if they're correct, they will correctly tell it, since the system is eventually synchronous) and use it for a time slot;
  - This still assumes the client knows how many nodes there are to wait for a valid quorum;
- **When a Byzantine node is discovered**: Add a firewall and block the sender's IP/port once Byzantine behaviour is detected coming from it.

References

[1] Google's gRPC Documentation. https://grpc.io/. Accessed: 2023/03/16.

[2] Google's GSON Documentation. https://github.com/google/gson. Accessed: 2023/03/16.

[3] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. *Department of Computing Science Technical Report Series*, 2001.

[4] Henrique Moniz. The Istanbul BFT consensus algorithm. *arXiv preprint arXiv:2002.03613*, 2020.